TECHNICAL FIELD

This invention relates to distributed computer systems, such as server data centers or Websites. More particularly, this invention pertains to a way to automatically enforce a policy in a multi-computer service application.

BACKGROUND

It is no secret that Internet usage has exploded over the past few years and continues to grow rapidly. People have become very comfortable with many services offered on the World Wide Web (or simply "Web"), such as electronic mail, online shopping, gathering news and information, listening to music, viewing video clips, looking for jobs, and so forth. To keep pace with the growing demand for Internet-based services, there has been tremendous growth in the computer systems dedicated to hosting Websites, providing backend services for those sites, and storing data associated with the sites.

One type of distributed computer system is an Internet data center (IDC), which is a specifically designed complex that houses many computers for hosting Internet-based services. IDCs, which also go by the names "Webfarms" and "server farms", typically house hundreds to thousands of computers in climate-controlled, physically secure buildings. These computers are interconnected to run one or more programs supporting one or more Internet services or Websites. IDCs provide reliable Internet access, reliable power supplies, and a secure operating environment.

Fig. 1 shows an Internet data center 100. It has many server computers 102 arranged in a specially constructed room. The computers are general-purpose computers, typically configured as servers. An Internet data center may be constructed to house a single site for a single entity (e.g., a data center for Yahoo! or MSN), or to accommodate multiple sites for multiple entities (e.g., an Exodus center that host sites for multiple companies).

The IDC 100 is illustrated with three entities that share the computer resources: entity A, entity B, and entity C. These entities represent various companies that want a presence on the Web. The IDC 100 has a pool of additional computers 104 that may be used by the entities at times of heavy traffic. For example, an entity engaged in online retailing may experience significantly more demand during the Christmas season. The additional computers give the IDC flexibility to meet this demand.

While there are often many computers, the Internet service or Website may only run a few programs. For instance, one Website may have 2000-3000 computers that run only 10-20 pieces of software. Computers can be added daily to provide scalability as the Website receives increasingly more visitors, but the underlying programs change less frequently. Rather, there are simply more computers running the same software in parallel to accommodate the increased volume of visitors.

Managing the physical resources of an Internet service is difficult today. Decisions such as when to add (or remove) computers to carry out functionality of the Internet service are made by human operators. Often, these decisions are made based on the operators' experience in running the Internet service. Unfortunately, with the rapid growth of services, there is a shortage of qualified operators who can make real-time decisions affecting the operation of a Website. Accordingly, it would be beneficial if some of the managerial, or policy aspects of running a Internet service could be automated.

Today, there is no conventional way to automate policy aspects of running an Internet service in a way that abstracts the functionality of the policy from the underlying physical deployment of the application. Perhaps this is because the industry has grown so fast that everyone's focus has been simply to keep up with the exploding demand by adding computers to Website applications. Not much thought has gone into how to model a policy mechanism that is scale-invariant.

At best, most distributed applications rely on human intervention and/or manual control for: (a) installing application components, (b) configuring the components, (c) monitoring the health of the overall application and individual components, and (d) taking reactive measures to maintain good overall application and component health. Where these tasks are automated, they operate in context of isolated physical components, without a big picture of how the components of the application relate to one another.

The downside with such traditional procedures to implementing policy is that Website operators must maintain constant vigilance over the operation of the application. Upon detecting a change in the application's operation that is contrary to the overall health of the application, Website operators are typically required to consult one or more documents that essentially detail the actions the Web Operators should take upon the occurrence of the condition. Moreover, such documents must continually be updated as the Website grows in physical resources and/or as the policy changes. Accordingly, it would be beneficial if some of the managerial, or policy aspects of running an Internet service could be automated.

SUMMARY

Described herein is a system and procedure to automatically enforce policy in distributed multi-computer service applications. Such service applications include multiple software programs that execute on multiple computers. The computers have access to communications media that allow data communications between different ones of the computers.

Multi-computer service applications are designed and implemented using a modeling system and a deployment system. The modeling system permits developers to architect the hardware and software used to implement the applications in an abstract manner. The modeling system defines a set of components used to describe the functionality of an application in a logical, scale-independent manner. In the described implementation, the modeling system defines several model components: a module, a port, and a wire; and a set of model extensions including, but not limited to: a store, an event source, an event sink, and an event path.

The module is the basic functional unit and represents a container of behavior that may be implemented by one or more computers running one or more software programs. For instance, in the context of a Website, one module might represent a front end that renders HTML pages, another module might represent a login database, and another module might represent a mailbox program. A port is a service access point for the module. All communications into and out of the module goes through a port. A wire is the logical binding that defines an allowed communication route between two ports.

The model components are arranged and interconnected to form a scale-independent model of the application. Each component specifies some functionality of the application.

Once a logical model is created, the deployment system uses the logical model to automatically deploy various computer/software resources to implement the application. The deployment system converts each of the model components into one or more instances that correspond to physical resources. As one example, the resources correspond to computer nodes of a distributed computer system that are loaded with specific types of software to implement the function represented by the model components. The deployment system initially installs the application and then dynamically and automatically modifies the resources used to implement the application in an ongoing basis as the operating parameters of the application change.

Each deployed module includes logical ports that are configured according to the logical model of the multi-computer service application. Each logical port is defined by port software. Logical data connections between the logical output and input ports are deployed according to the logical model. Each port is configured to communicate through different numbers of logical data connections without modifying the port software.

A deployed module, upon the occurrence of a condition, sends a notification to a policy module. The policy module responds to the notification by formulating a request for one or more destination modules according to a policy. The policy module implements one or more policies devised by the developer or operator of the service application. The policy module provides the request to an output port of the policy module. The output port forwards the request to input ports of multiple modules according to the configured logical data connections.

Upon instantiation, or deployment, the output ports are configured to specify different logical data connections. Output ports can also be configured during run-time to specify different logical data connections. In either case, the output ports forward notifications to multiple modules and input ports in accordance with the logical connections specified for said particular output port.

In one implementation, the policy module monitors the operation of the multi-service computer application during runtime. The monitored operations are evaluated against a policy. The policy module determines the number of instances of each module used to implement the multi-computer service at any given time based on the policy.

The policy module responds to changes in operating conditions by automatically specifying an action selected from a group of actions. Such actions include (a) deploying a new resource represented by a model component in the logical model, (b) manipulating a module in multi-service computer application by sending events to the module, and (c) removing a module from the multi-service computer application.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 illustrates a conventional Internet data center (IDC).

Fig. 2 illustrates a set of model components that form the building blocks for modeling an application, along with the associated schema.

Fig. 3 illustrates a database application for an IDC that is modeled in terms of the components.

Fig. 4 illustrates an Internet-based email application for an IDC.

Fig. 5 is a block diagram of a computer that may be used to implement the modeling software for modeling the IDC.

Fig. 6 is a flow diagram of a process for modeling an IDC.

Fig. 7 is a block diagram of a deployment system that converts a logical model to a fully functioning physical implementation

Fig. 8 illustrates a translation of the logical model into real-world instances.

Fig. 9 illustrates exemplary data records of an instance database used to store the real-world instances.

Fig. 10 is a flow diagram of a process for deploying resources for the application based on the logical model.

Fig. 11 illustrates a system providing automatic policy enforcement in a multi-computer application.

Fig. 12 illustrates a procedure that provides automatic policy enforcement in a multi-computer application.

DETAILED DESCRIPTION

A system and procedure to automatically enforce policy in a multi-computer service application includes a modeling system and a deployment system. The modeling system permits developers of applications for distributed computer systems (e.g., server data centers, Internet data centers (IDCs), Web farms, and the like) to architect the hardware and software in an abstract manner. The modeling system defines a set of components used to describe the functionality of an application in a logical, scale-independent manner. An "application" within this context refers to an entire service hosted on the distributed computer system.

For instance, an Internet data center may host a Website for an online retailer, where the application entails the entire software and hardware configuration that implements the online retailer's Internet presence. The application might include, for example, a front end to handle client requests, an order processing system, a billing system, an inventory system, a database system, and a policy module to respond to changes in the operation of the application.

The model components are arranged and interconnected to form a scale-independent model of the application. Each component specifies some functionality of the application. The model can then be used to construct a scalable physical blueprint in terms of which machines run which pieces of software to form the application.

The deployment system uses the logical model to deploy various computer/software resources in real-time as the applications need them. The deployment system converts each of the model components into one or more instances that correspond to physical resources. The deployment system tracks the instances and all available resources. The deployment system decides when resources should be added (or removed) and monitors the current state of implementation. The deployment system installs the application and then dynamically and automatically modifies the resources used to implement the application in an ongoing basis as the operating parameters of the application change.

One of the components deployed by the deployment system is a policy module. The policy module is one of the model components in the logical model. The policy module responds to changes in the operation of the application by enforcing a policy. The policy module receives notifications from the deployed resources. In response to receiving such notifications, the policy module evaluates the notifications against a policy, which results in one or more of data, a request, or notification being sent to one or destination resources according to the configured logical data connections.

The system is described in the context of Website designs, such as applications for Internet data centers. However, the design system may be implemented to model other large size and scalable applications for computer systems. Accordingly, the design system can be implemented in a wide variety of ways, including Internet-based implementations and non-Internet-based implementations.

Model Components and Schema

The modeling system defines several model components that form the building blocks of a logical, scale-independent application: a module, a port, and a wire. It also defines a set of model extensions including, but not limited to: a

10

store, an event source, an event sink, and an event path. In a design tool, the components are represented pictorially as graphical elements or symbols that may be arranged and interconnected to create scale-independent models of Website applications. The graphical elements have an associated schema that dictates how the functional operations being represented by the graphical elements are to be specified.

Fig. 2 illustrates a set of model components 200 that a module, as represented by modules 202(A)-202(C), a store 204, ports 206, wires 208, event sources 210, event sinks 212, and event paths 214. The components 200 are arranged in a no particular manner other than to foster discussion of their individual traits.

A module 202 represents a basic unit of functionality for the application and is depicted as a block. It is a logical entity that represents some portion of the application as implemented at the IDC, but has no physical manifestation. The module often corresponds to a software program that handles a logical set of tasks for the application. For instance, one module might represent a front end for a Website, another module might represent a login database, and another module might represent an electronic mail program.

Each module 202 is a container of behavior. A simple module is atomic and has associated a unique identifier. Modules can be nested into a hierarchy of modules to form more complex behaviors. In a module hierarchy, the leaf modules are simple modules, and the non-leaf modules are compound modules.

Each module 202 defines a unit of scaling. While one module logically represents a functional operation of the application, the module may translate to any number of computers when actually implemented. In this way, the module is scale-independent, allowing the number of underlying computers used to implement the module to change at any time. When converted to a physical implementation, "instances" are created from the modules. The module instances are assigned a unique identifier and maintain ancestral data regarding which module created them. The instances of simple modules are called "engines", which correspond to software programs that run on individual computer nodes.

A store 204 is the most basic unit of storage and is depicted graphically as a disk storage icon. It represents a logical partitioning, which may be implemented by any number of physical disks or other storage media.

A port 206 is a service access point for a module 202 or store 204 and is depicted as spherical knobs projecting from the module or store. All service-related communications into and out of the module go through the port 206. Each port 206 has a "type", which is a set of attributes describing format, semantics, protocol, and so forth. At runtime, the port represents a set of physical ports associated with the instantiated engines of the modules.

A wire 208 is the logical binding that defines a communication route between two ports 206 and is depicted as a bold line. Each wire 208 can be type-checked (i.e., protocols, roles) and defines protocol configuration constraints (e.g., HTTP requires TCP, TCP requires IP, etc.).

Event sources 210 and event sinks 212 are used for discrete semantic messaging between modules and stores. An event source 210 is pictorially shown as a triangle pointing away from the module or store, while an event sink 212 is depicted as a triangle pointing toward the module or store. An event path 214 is a logical connection between sources and sinks, and carries event messages used to inform modules/stores and implement policy (e.g., scaling, fail-over, monitoring, application processes, etc.). It is depicted as a dashed line.

Fig. 2 also illustrates the schema underlying the graphical elements as exemplary data structures associated with the model components. Module 202(A) has an associated structure 220 that contains various characteristics for the module, such as functionality, processing requirements, software, and so forth. Thus, a module for a database server function might have characteristics pertaining to the kind of database (e.g., relational), the data structure (e.g., tables, relationships), software (e.g., SQL), software version, and so forth. Modules 202(B) and 202(C) have similar structures (not shown).

The store 204 has a corresponding structure 222 that defines the requirements for storage. The store schema structure 222 might include, for example, the kind of storage (e.g., disk), the storage format, and so on. Each port 206 has a schema structure, as represented by structure 224, which dictates the port's type. Each wire 208 also is also associated with a structure, such as structure 226, which outlines the protocols implemented by the connection. Similar schema structures may also be provide for event sources, event sinks, and paths.

Using the model components, an application developer can logically configure scale-independent applications prior to physically laying them out in the Internet data centers. The developer drafts a model using a user interface to select and interconnect the model components. Once constructed, the modeling software generates the application based on the depicted model and the underlying schema. The application may subsequently be converted into a physical blueprint that details the computers and software needed to implement the application for a specified number of client visitors.

The scale-invariant nature of the modeling system allows application developers to focus only on designing software for a specific functional task (e.g., front end, login database, email program, etc.). All external communications can then be expressed in terms of transmitting to and receiving from one or more associated ports. In this manner, the application developers need not worry about how many machines will be used to run the module, or how other modules of the scale-independent application are being configured.

Exemplary Module and Application

Fig. 3 shows a fault-tolerant SQL (structure query language) database module 300 to demonstrate how the model components may be organized and connected to represent a portion of an application. In this example, the database module 300 represents a SQL database that may be used independently or as a component in a larger application. The SQL database module 300 has a module interface composed of a single port 302 that implements a TDS (Tabular Data Stream) protocol.

The SQL database module 300 is a compound module made up of three simple modules: a fail-over policy module 310, a primary SQL module 312, and a secondary SQL module 314. The primary and secondary SQL modules represent dual programs that operate in parallel so that, in the event that the primary module 312 crashes, the secondary module 314 can assume the role without loss of service. The database module 300 also has a data store 316 that represents the memory storage for the SQL database module.

The primary SQL module 312 has a module interface that includes a first port 320 for communicating with the compound module port 302 and a second port 322 for communicating with the store 316. The primary SQL module 312 also has an event source 324 and an event sink 326 for handling event messages from the fail-over policy module 310. Similarly, the secondary SQL module 314 has a module interface with a first port 330 for communicating with the compound module port 302, a second port 332 for communicating with the store 316, and an event sink 334 for receiving events from the fail-over policy module 310. A wire 336 interconnects the external compound module port 302 with the ports 320 and 330 of the primary and secondary SQL modules, respectively.

The store 316 has a port 340 to communicate with the primary and secondary SQL modules 312 and an event sink 342 to receive event messages from the fail-over policy module 310. A wire 344 interconnects the store port 340 with the ports 322 and 332 of the primary and secondary SQL modules, respectively.

The fail-over policy module 310 has a module interface that includes three event sources and one event sink. An event sink 350 receives a "fail" event from the event source 324 of the primary SQL module 312 via an event path 352 when the primary SQL module experiences some failure. In response to receiving a "fail" event, the fail-over policy module 310 concurrently issues a first event to stop the failed primary module 312, another event to assign the secondary module 314 as the new owner of the store 316, and a third event to start the secondary module 314. The "stop" event is issued via an event source 354 over an event path 356 to the event sink 326 of the primary SQL module 312. The "stop" event directs the primary SQL module 312 to halt operation.

The fail-over policy module 310 issues an "assign owner" (AO) event from an event source 358, over the event path 360 to the event sink 342 of the store 316. The assign owner event directs the storage mechanisms to switch to allowing access by the secondary SQL module 314, rather than the primary SQL module 312. The fail-over policy module 310 also issues a "start" event from event source 362 over event path 364 to the event sink 334 of the secondary module 314. The start event directs the secondary SQL module to start operation in place of the primary SQL module.

The SQL database module 300 illustrates how the model components—modules, stores, ports, wires, sources, sinks, and paths—may be arranged and interconnected to form a complex module. The developer specifies the characteristics associated with each component according to the prescribed schema. The complex module may in turn be added to other simple or complex modules to form other complex modules. Eventually, the largest complex module becomes the application, which may then be used to form a blueprint for constructing the data center.

Fig. 4 shows a simplified application 400 for an online retailer. The application 400 includes a front end module 402, a catalog module 404, an order processing module 406, and a fulfillment module 408. The application 400 also includes a customer database 410 and the fault-tolerant SQL database module 300. Notice that the SQL database module 300 is the same as that shown in Fig. 3 to illustrate how complex modules can be nested into even greater complex modules to form an application.

The front end module 402 handles requests from clients who wish to shop with the online retailer. The front end module 402 has a port 420 that accommodates communications with external clients using the TCP/IP protocol over the Internet. The front end module 402 also has an order port 422 to define a communication exchange with the order processing module 406 and a catalog port 424 for communication flow to the catalog module 404. The ports 422 and 424 may be configured according to any of a variety of types, which support any one of a number of protocols including SOAP, TCP, or UDP. An event sink 426 is also provided to receive a "new product" message from the catalog module 404 when a new product has been added to the catalog.

The catalog module 404 provides catalog information that may be served by the front end to the requesting clients. The catalog module 404 has a front end port 430 connected via a wire 432 to the catalog port 424 of the front end module 402. The front end port 430 has a type that matches the catalog port 424. The catalog module 404 also has an event source 434 that for communicating the "new product" messages over path 436 to the event sink 426 of the front end module 402.

A SQL port 438 interfaces the catalog module 404 with the SQL database module 300. The SQL port 438 has a type that utilizes the TDS protocol for the communication exchange with the external port 302 of the SQL database 300.

The order processing module 406 has a front end port 440 to define a communication interface with the front end module 402 via a wire 442. The order processing module 406 also has a fulfillment port 444 to facilitate communication with the fulfillment module 408 over wire 446 and a database port 448 to facilitate communication with the customer database 410 via wire 450.

An event source 452 is provided at the order processing module 406 to pass "order complete" events to the fulfillment module 408 via path 454. These events inform the fulfillment module 408 that an order is complete and ready to be filled. A second event source 456 passes "new account" events to the customer database 410 via path 458 whenever a new customer orders a product.

The fulfillment module 408 has an order port 460 to provide access to the wire 446 to the order processing module 406 and a database port 462 to interface with the customer database 410. The fulfillment module 408 also has an event sink 464 to receive the "order complete" events from the order processing module 406.

The customer database 410 has an order port 470 to provide access to wire 450 and a fulfillment port 472 to facilitate communication with the fulfillment module 408 via wire 474. The customer database 410 further has an event sink 476 to receive the "new account" events from the order processing module 406.

The modeling approach illustrated in Figs 3 and 4 is tremendously beneficial because it allows developers and IDC operators to view the entire operation in terms of abstract functional pieces that are scale-independent. The online retailer application 400, for example, requires a front end unit, a catalog unit, an order processing unit, and a fulfillment unit regardless of whether the retailer is handling 100 hits a day or 10 million hits per day.

The scale-independent nature frees the developer to focus on his/her little piece of the application. For instance, a developer assigned the task of building the front end module 402 need only be concerned with writing software code to facilitate response/reply exchanges. Any communication to and from the module is defined in terms of order-related data being passed to the order processing module 406 via the order port 422 and product data being received from the catalog module 404 via the catalog port 424. The developer defines the data flow to and from the order port 422 and the catalog port 424 according to their respective associated protocol types.

The application 400 can then be used to construct a computer system that hosts the online retailer. Initially, the online retailer may not receive very much traffic, especially if launched away from the Christmas season. So, perhaps the front end module 402 translates initially to only a few computers to handle the light traffic from the Internet. But, suppose that over time the site becomes more popular and the Christmas season is fast approaching. In this situation, the online retailer may authorize the IDC operator to add many more computers for the front end tasks. These computers are equipped the software and configured to accept HTTP requests for product information and to serve Web pages containing the product information. The computers are added (or removed) as needed, without altering the basic application 400.

Computer-Based Modeling System and Method

Fig. 5 shows an exemplary computer system 500 that implements modeling software used to design Website applications. The modeling computer may be implemented as one of the nodes in a Website, or as a separate computer not included as one of the nodes. The modeling computer has a processor 502, volatile memory 504 (e.g., RAM), and non-volatile memory 506 (e.g., ROM, Flash, hard disk, optical, RAID memory, etc.). The modeling computer 500 runs an operating system 510 and modeling system 512.

For purposes of illustration, operating system 510 and modeling system 512 are illustrated as discrete blocks stored in the non-volatile memory 506, although it is recognized that such programs and components reside at various times in different storage components of the computer 500 and are executed by the processor 502. Generally, these software components are stored in non-volatile memory 506 and from there, are loaded at least partially into the volatile main memory 504 for execution on the processor 502.

The modeling system 512 includes a user interface 514 (e.g., a graphical UI) that presents the pictorial icons of the model components 516 (e.g., modules, ports, sources, sinks, etc.), a component schema database 518, and a logical-to-physical converter 520. The modeling system 512 allows a developer to design an application for an IDC by defining modules, ports, and event message schemes. The user interface 514 presents symbols of the components 516, such as the symbols shown in Figs 2-4, and permits the developer to arrange and interconnect them. The UI 514 may even support conventional UI techniques as drag-and-drop operations.

The symbols depicted on the screen represent an underlying schema 518 that is used to define the model. For instance, a block-like module symbol is associated with the characteristics of the functionality that the module is to represent in the application. Thus, the developer may define a database module that has characteristics pertaining to the kind of database (e.g., relational), the data structure (e.g., tables, relationships), software (e.g., SQL), software version, and so forth. Accordingly, by drafting the model on the UI, the developer is architecting the entire schema that will be used to design the scale-independent application.

Once the application is created, the logical-to-physical converter 520 converts the application to a physical blueprint that details the number of computers, software components, physical ports, and so forth. The converter takes various parameters—such as how many site visitors are expected, memory requirements, bandwidth requirements, processing capabilities, and the like—and scales the application according to the schema 518 represented by the application. The converter 520 specifies the number of computers needed to implement each module, the number of disks to accommodate the stores, and the types of communications protocols among the modules and stores.

In this manner, the modeling system changes the development effort from a node-centric approach to architecting IDCs to an application-centric approach. Within conventional node-centric methodology, the focus was on the computers and how they were laid out. The application was then loaded onto the nodes in a somewhat ad hoc manner. With the new application-centric approach, the focus is initially on the application itself. The physical nodes used to implement the application are derived once the application is specified.

Fig. 6 shows a method for modeling a scale-independent application for an Internet data center. The method 600 may be implemented, for example, by the modeling system 512 executing on the modeling computer 500. In such an implementation, the method is implemented in software that, when executed on computer 500, performs the operations illustrated as blocks in Fig. 6.

At block 602, the modeling system 512 allows the developer to define the modules and stores that form the functional elements of the applications. The UI 514 enables the developer to create modules and stores and to define their characteristics as prescribed by a predetermined schema. This entry process begins to construct the logical building blocks of the application.

At block 604, the modeling system 512 enables the developer to define the ports for the modules and stores. The developer selects the type of ports and ensures compatibility of ports that are connected to one another. At block 606, the

21

developer also defines any events that may be passed among modules and stores. The developer creates event sources and event sinks to accommodate the various events. At block 608, the developer uses the modeling system 512 to interconnect the ports with wires and the event sources/sinks with event paths. By joining the various modules and stores, the developer effectively forms a logical representation of the application.

At block 610, the modeling system 512 generates an application using the graphical representation constructed by the developer. The modeling system 512 generates the logical specifications associated with the graphical model, including the characteristics of the modules and stores as well as the types of the ports and event sources/sinks. The application provides a complete logical representation of the service that will eventually be implemented at the Internet data center. The application may be stored on disk or some other form of computer-readable medium (block 612).

At block 614, the modeling system 512 converts the application to a physical blueprint that specifies the computers, the software run by each of the computers, and the interconnections among the computers. This physical blueprint may be used by the operator to install and manage the service afforded by the application.

Computer-Based Deployment System

Once a logical model is created, an automatic computer-based deployment system uses the logical model to deploy various computer/software resources to implement the application. The deployment system converts each of the model components into one or more instances that correspond to physical resources, such as nodes of a distributed computer system that are loaded with specific types of software to implement the function represented by the model components. The deployment system initially installs an application on the physical resources according to the logical model. It then dynamically and automatically modifies the resources used to implement the application in an ongoing basis as the operating parameters of the application change.

Fig. 7 shows a deployment system 700 that converts the logical model to a fully functioning physical implementation. The deployment system 700 includes a policy module 702, a core runtime logical-to-physical converter 704, and hardware/software resources 706 that are all interconnected via a wireless and/or wire-based communication network 708 (e.g., a LAN, a WAN, intranet, Internet, combinations thereof, etc.). In this example, the hardware/software resources are illustrated as computer nodes of a distributed computer system, as represented by computers 706(1), 706(2), ..., 706(N). The policy module 702 and core runtime converter 704 may be implemented on one or more computers, which may or may not be part of the nodes in the distributed computer system.

For purposes of discussion, the deployment system 700 is described in the context of an Internet service that is executed at an Internet data center having an abundance of generic computer nodes. The nodes can be allocated to one or more Internet services from a reserve pool of nodes, as illustrated in Fig. 1.

The policy module 702 implements one or more policies devised by the developer or operator of the Internet service. The policies specify when instances derived from the logical model should be created, manipulated, and destroyed. The policy module monitors various events generated by the nodes and implements policy decisions regarding how to handle the events. By specifying when and what instances should be created (or destroyed), the policy module 702 effectively dictates when hardware/software resources 706 should be added (or removed) to support the changing demands of the Internet service.

The core runtime converter 704 implements the policy decisions made by the policy module 702. The runtime converter 704 has a service running state 710 that tracks all instances of the model components currently in existence. That is, the service running state 710 tracks the elements in the physical world with respect to the logical model. The service running state 710 maintains a copy of the logical model, such as online retailing model 400 of Fig. 4. The logical model is created by the modeling system described above with respect to Figs. 5 and 6. The current instances are maintained in a database 714. The records in instance database 714 include such information as identify of the instance, the name of the logical component from which it is derived, the node on which it is running, the network addresses representing the ports of the modules and stores, type of software loaded on the node, various protocols supported by the instance, and so forth.

The instances are derived from the logical model. The policy module 702 articulates the number of instances of each model component used to implement the Internet service at any given time. For instance, suppose the Internet service requires one hundred computers to effectively implement a front end that handles site traffic at 99.9% efficiency with each computer running at 70% utilization. The policy module might further specify that more computers should be added if some policy threshold is met (e.g., efficiency rating drops below some threshold or computer utilization rises above some threshold) or removed if another threshold is met.

A resource manager 716 tracks all of the physical resources available to the Website. These resources include computer nodes, storage, software, and so forth. Records identifying all of the resources are kept in the resource database 718. For instance, there might be one record for each computer node, storage device, and software module in the Internet data center. The records contain such information the identity of the allocated nodes, computing characteristics and capabilities, the application(s) to which they are allocated, the date and time of allocation, and so forth.

The resource manager 716 allocates the resources as needed or requested by the Internet service according to the policy implemented by the policy module 702. The allocation depends upon the availability of resources at the time of request. The resource manager 716 may also recover resources that are no longer needed by the Internet service and return the resources to the pool of available resources.

Upon allocation (or recovery) of a resource, the resource manager 716 post a record to the resource database 718 reflecting which resource is allocated to (or recovered from) which application. As an example, when an Internet service desires more nodes for the front end tasks, the resource manager 716 allocates one or more free nodes from the pool of resources to the Internet service.

Another unit, referred to as "new" 720, manages the creation of new instances in the physical world from the model components specified in the logical model 400. A loader 722 carries out the configuration of newly allocated resources to the functions dictated by the new instances. In this manner, when another instance of a component is desired, the new program 720 communicates with the resource manager 716 to allocate a node from the resource pool and with the loader 722 to load the appropriate software programs onto the node.

Each computer node 706(1)-706(N) has a policy engine 730(1)-730(N) and a node loader 732(1)-732(N). The policy engine 730 locally implements the policies dictated by the policy module 702. Depending upon the policies, the policy engine 730 monitors various parameters and performance metrics of the local node and generates events that may be sent to inform the policy module 702 as to how the node is operating within the policy framework.

For instance, suppose a policy specifies that nodes used to instantiate a particular module of the logical model should not operate at more than 80% utilization. The policy engine 730 enforces this policy by monitoring utilization. If utilization rises above 80%, the policy engine 730 sends an event to the policy module 702 to notify of the utilization increase. With this information, the policy module 702 may decide to create another instance of the module to relieve the over-utilized node.

The node loader 732 performs the actual loading tasks specified by the loader 722 in the core runtime converter 704. It is the local code on an otherwise generic node to which the runtime loader 722 may communicate when configuring the node.

Fig. 8 shows an example of a portion of logical model 400 being converted into actual instances. To illustrate the conversion, the front end module 402 and the order processing module 406 are extracted from the online retailer service application 400 (Fig. 4). A wire 442 interconnects the two modules by logically coupling the ports 422 and 440.

26

The front end module 402 in the logical model translates to one or more computer nodes in the physical world that runs software for handling client queries. These physical instances are represented diagrammatically by the rectangles with ovals. Here, based on a policy, the front end module 402 is converted into multiple front end instances 800(1), 800(2), 800(3), ..., 800(J), which each corresponds in one-to-one fashion with a computer node loaded with software used to implement the front end of the service. The order processing module 406 translates to one or more computer nodes that run a program for processing client orders. In Fig. 8, the order processing module 406 is converted into plural order processing instances 802(1), 802(2), ..., 802(K).

The ports 422 and 440 represent protocols and roles within protocols in the logical world. They translate to the physical world as a set of network addresses used to communicate with the software at the respective modules. For instance, the physical translation of a logical port might be IP Port 80, using HTTP (hypertext transport protocol). In Fig. 8, one logical port in the logical model is converted to a port address for each instance of the module. Logical port 422 converts into physical address ports 804(1)-804(J) and logical port 440 converts into physical ports 806(1)-806(K). Furthermore, the logical ports might correspond to particular objects and their interfaces. For example, port A might correspond to an object A and interface on a first computer, while port B might correspond to an object B and interface on a second computer.

The wire 442 represents logical connections between ports. It translates to a physical mesh of all possible communication paths between the instances of each module. The number of wires is determined as the cross product of the number of instances of each module. In Fig. 8, the wire 442 converts to a physical connection between every instance 800(1)-800(J) of the front end module 402 and every instance 802(1)-802(K) of the order processing module 406.

Fig. 9 illustrates exemplary records 900 in the instance database 714 that track the instances derived from the logical model 400. In this example, the database is a relational database that stores records in tables, and the records may be linked to one another via relationships. Here, there are three tables: a module table 902, a port table 904, and a wire table 906. Each table holds one record for a corresponding instance of the logical model. Each record has a number of fields relating to the type of information that is being tracked for each instance.

The module table 902 tracks instances of modules in the logical model. There is one record for each module instance. Thus, with respect to the front end module of Fig. 8, there are "J" records in the module table 902 corresponding to the "J" front end instances 800(1)-800(J). Each record in the module table 902 contains an instance ID to identify the individual instance, an identity of the module component from which the instance is derived, a node ID to identify the computer node to which the instance is associated, the type of software loaded on the node to implement the module functionality, a software ID, an identity of the various ports to the modules, and various protocols supported by the module instance. It is noted that other implementations may include additional fields or fewer fields than those illustrated.

The port table 904 tracks instances of the port in the logical model, such as ports 422 and 440 in Fig. 8. A record from this table includes such information as the port ID, the model component identity, a node ID, the network address represented by the port, the instance ID of the corresponding instance, the protocol

28

used by the port, and an ID of the wire to which the port is connected. Again, more or less fields may be used in other implementations.

The wire table 906 tracks instances of the wires in the logical model, such as wire 442 in Fig. 8. A record in the wire table includes a wire ID, a model component identity, a node ID, a port ID, an instance ID, and the protocol supported by the wire.

Notice that the three tables can be correlated with one another via various relationships. For example, the module table 902 and the port table 904 are related by various data fields, such as the port ID field and the instance ID field. The wire table 906 correlates with the port table 904 via wire ID and port ID and with the module table 902 via instance ID. Notice also that the tables have a node ID field that provides a reference into the resource database by identifying which node the instance is associated with.

It is noted that the illustrated arrangement of the database is merely for discussion purposes. Many other table arrangements with more or fewer tables than illustrated.

Realtime Deployment Method

Fig. 10 shows a method 1000 for deploying resources for an application based on the logical model. The method 1000 is implemented by the deployment system 700 of Fig. 7 and hence can be embodied as software that, when executed on one or more computers, performs the operations illustrated as blocks in Fig. 12.

At block 1002, the policy module 702 monitors various operating parameters and listens for events from the policy engines 730 at the individual nodes 706. The policy module 702 evaluates the parameters and events against

29

the policy backdrop to determine whether the current physical implementation is satisfactorily supporting the application, or whether a new instance of some component should be added (block 1004). It is noted that the continuing process is described in the context of adding a new instance. However, the policy may alternatively dictate that one or more instances should be removed. Removal of instances is somewhat easier in that instances are deleted from the instance database 714 and the computer nodes are returned to the additional pool for reallocation.

Assuming that a new instance is desired (i.e., the "yes" branch from block 1004), the policy module 702 consults the service running state 710 to understand the current number and arrangement of instances (block 1006). The policy module 702 can request various types of information of the service running state 710, such as:

How many instances of a given module?
What nodes are associated with a given model component?
Which nodes are attached to a given wire?
What is the network address of a node?
What wire is attached to a port on a given component?
What components does a given component own?

Depending upon the event or operating condition, the policy module 702 can request information on a particular module in the logical model 400. For example, assume that an event has been received from a front end node indicating that the utilization has risen to above 90%. In response, a policy specifying the addition of another instance at such utilization levels is triggered. The policy module 702 asks the service running state 710 how many instances of the front end module currently exist, and information regarding how to specify an instance for the front end module.

At block 1008, the policy module 702 calls the program "new" 720 to create a new instance of a module in the logical model. At block 1010, the new program 720 calls the resource manager 716 to request allocation of a new node (assuming one is available). The resource manager 716 examines the resources using data from the resource database 718 and allocates a new node that is currently available from a pool of free nodes. The resource manager 716 records the allocation in the resource database 718, identifying which node is allocated, what application it is being allocated to, the date and time that it is allocated, and so forth.

At block 1012, the new program 720 calls the loader 722 to install software onto the allocated node and configure the node to perform the functions represented by the logical module from which the instance is created. In response, the loader 722 initializes the node by communicating with the node loader 730 of the new node via the network 708 to install an image of an operating system (e.g., Windows NT server operating system from Microsoft Corporation). The loader 722 then loads the appropriate software and configures the node to perform the functions represented by the logical model component from which the instance is derived. For example, for an instance of an SQL module in the logical model 400, the node loader 732 loads SQL server software. The loader 722 registers the physical port addresses with the service running state 710, which records the new instance in the instances database 714.

At block 1014, the service running state 710 is notified when the newly allocated and configured node is up and running. The service running state 710 records a new instance of the logical model in the instances database 714. The record reflects an ID of the instance, the name of the logical component from which it is derived, the allocated node, the network addresses of the node, software type and IDs, various protocols supported by the instance, and so on.

At block 1016, the policy module 702 is notified by the core runtime converter 704 that a new instance is up and running. The new instance should relieve the event or operating condition to bring operation back into compliance with the policy.

Policy Enforcement Mechanism

Fig. 11 illustrates a system 1100 to enforce policy in a multi-computer service application having a plurality of software programs that execute on a plurality of computers. The service application includes a communications medium that allows data communications between different ones of the computers. In this example, the application is a distributed copying service, where copy modules 1102 send electronic documents to front ends 1104.

The application is designed and implemented using the modeling system and deployment system discussed above. The model components of the application include the logical counterparts of the policy module 1101, one or more copy modules 1102, and one or more front ends 1104. The modeling system allows the model components to be arranged and interconnected to form a scale-independent model of the application. The deployment system converts each of the model components into one or more instances that correspond to physical resources. In this example, the physical instances of the system's model components are represented diagrammatically by rectangles with ovals.

Each module includes respective logical input ports 1106 and logical output ports 1108. The logical input ports 1106 and logical output ports 1108 are configured on different modules in accordance with a logical model 1124 of the distributed copying service. Each logical input and output port is defined by port software.

Logical wires or data connections are configured between the logical output and input ports in accordance with the logical model. For example, data flow arrows 1110, 1112, and 1114 represent logical data connections. Each module can include other communication paths other than those represented by the data flow arrows. Upon each module's instantiation, each respective port is configured to communicate through different numbers of logical data connections without modifying the port software.

The policy module 1101 implements one or more policies devised by the developer or operator of the service application. The policy module monitors various events generated by the computer nodes and implements policy decisions regarding how to handle the events. In this example, the front end module 1104 represents a newly instantiated module that is ready to receive electronic data from copy modules 1102. The front end module provides a notification 1110 to output port 1108-2, which is configured according to the configured logical data connections to send the notification to the policy module's input port 1106-1. The policy module accepts the notification and in response, formulates a request for one or more destination modules. As noted above, such a request includes, data or notifications.

The request is provided to the output port 1108-1 of the policy module. The output port forwards the request 1112 to respective input ports 1106 of one or more copy modules in accordance with the configured logical data connections. In this example, the request notifies the copy modules that they can begin sending data to the newly instantiated front end module.

In response to receiving the request, the respective output ports 1108-3 through 1108-N are reconfigured, during runtime, according to the logical connections 1114 described in the logical model to send data to the new front end.

In this manner, the policy enforcement system changes distributed application management from a manual process requiring human intervention and control, to an automatic process. With conventional management procedures, a sending module would have had to have been recoded with the new addresses of any added components. With the new automatic policy enforcement, scale-invariant policies can be developed that do not require constant human intervention and control to respond to changes in application operation.

Fig. 12 illustrates an exemplary procedure 1200 to automate policy enforcement in a multiple-computer application. At step 1202, the procedure configures logical output ports and logical input ports on different modules according to a logical model (logical model) of the multi-computer service application. Each logical input and output port is defined by port software.

At step 1204, the procedure configures logical data connections between the logical output and input ports according to the logical model. At step 1206, the procedure configures each port to communicate through different numbers of logical data connections. This is done without modifying the port software.

At step 1208, upon occurrence of a condition, the procedure sends a notification from a particular module to a policy module. At step 1210, the policy module receives the notification. At step 1212, the policy module responds to the notification by determining a request for one or more destination modules. This request is determined based on a policy developed by the designer of the application. At step 1214, the policy module provides the request to an output port of the policy module to forward the request to input ports of multiple modules according to the configured logical data connections. In this way, the response to the notification is automatically forwarded to the proper modules without a need to determine who or where those modules are.

Conclusion

Although the description above uses language that is specific to structural features and/or methodological acts, it is to be understood that the invention defined in the appended claims is not limited to the specific features or acts described. Rather, the specific features and acts are disclosed as exemplary forms of implementing the invention.